EVA: Enhanced Virtual Assistant for Intelligent System Control

Mrs. Ashvini V[1], G Vikas[2], Pola Vikram Teja[3]
[1]*Assistant Professor, Department of CSE (Data Science), Vemana Institute of Technology, Bengaluru, India*
[2,3]*Student Department of CSE (Data Science), Vemana Institute of Technology, Bengaluru, India*

***Abstract*—EVA an intelligent multimodal desktop assistant simplifies computer interaction through natural voice commands, visual awareness, persistent memory, and automated task execution. The system can understand user requests, observe relevant screen and camera information, manage applications and files, and maintain contextual continuity across sessions. A graphical interface provides clear feedback and allows users to monitor or interrupt ongoing operations. Safety mechanisms are incorporated to restrict unauthorized file access, protect sensitive information, and verify the outcome of automated actions. The developed system demonstrates the potential of multimodal artificial intelligence to provide responsive, reliable, and user-controlled assistance for everyday desktop activities.**

***Index Terms*—Virtual Assistant, Desktop Automation, Multimodal AI, Voice User Interface, Human–Computer Interaction, System Control**

I. INTRODUCTION

The proliferation of large language models (LLMs) and realtime multimodal streaming APIs has enabled AI systems to move beyond passive question-answering toward interactive assistance on personal computers. Commercial voice assistants and browser-based LLM interfaces provide broad conversational capabilities, but generally offer limited continuous awareness of the user's screen, local files, and ongoing desktop tasks. Moreover, an assistant that can invoke operating-system actions must distinguish a generated statement of success from a verified execution result.

EVA addresses these gaps by combining (i) low-latency bidirectional audio streaming, (ii) real-time screen and webcam perception, (iii) a Fast-Path Intent Router for local commands, (iv) grounded operating-system and application handlers, (v) zero-dependency vector retrieval across sessions, and (vi) a dedicated safety and hardening layer. The result is a multimodal desktop presence capable of hearing, seeing, remembering, and acting with verifiable outcomes.

EVA is an interruptible voice pipeline with screen and webcam perception, verified operating-system and document automation, cross-session semantic memory, and contextaware assistance. It also enforces subprocess isolation, path containment, and action grounding to prevent false reports of task completion.

II. LITERATURE REVIEW

A review of recent intelligent assistants, voice interfaces, multimodal systems, and desktop-automation methods was conducted to identify their capabilities and remaining research gaps.

A. Review of Related Systems

1. NAIA: A Multi-Technology Virtual Assistant for Boosting Academic Environments—Mendoza et al., 2025. This work presents a multimodal AI virtual assistant integrating LLM, computer vision, voice processing, and digital avatars for academic support. Advantages: Supports multiple roles, voice interaction, visual awareness, and task automation. Disadvantages: Mainly designed for academic environments and depends on external AI services.
2. Machine Learning-Enhanced Voice Interaction: Revolutionizing Windows Desktop Applications—Vora et al., 2025. This research focuses on improving voice interaction in Windows desktop applications using CNNs, RNNs, and Transformers. Advantages: Improves speech accuracy, speaker

adaptability, and real-time response. Disadvantages: Challenges remain in handling noise, privacy, multilingual support, and contextual understanding.

3. Development of ALEN: An Advanced Voice Assistant with Comprehensive Functionalities—Pundir et al., 2025. This work presents a Python-based voice assistant integrating automation, multimedia control, web browsing, application control, and WhatsApp automation. Advantages: Modular, scalable, supports system control and privacy through password protection. Disadvantages: Limited contextual understanding and unreliable speech recognition in noisy environments.
4. Developing Voice-Based Assistant: Machine Learning Techniques to Enhance Learning Experiences for Students with Special Needs—Fadel et al., 2025. This work develops a voice assistant using ASR, NLP, and a fine-tuned LLM for supporting visually impaired students. Advantages: Provides personalized, context-aware, and hands-free learning support. Disadvantages: Focused mainly on Python and machine learning education for visually impaired learners.
5. Context-Aware AI Assistant Using Open AI—Kavitha et al., 2025. This work focuses on developing a contextaware personal AI assistant using NLP and machine learning to understand user intent and preferences. Advantages: Improves personalization and maintains conversational context. Disadvantages: Long-term context handling and integration of information from multiple sources remain challenging.
6. Artificial Intelligence for Speech Classification and Enhancement of Speech and Language Disorders—Remya et al., 2025. This research reviews AI and deep learning techniques for speech classification and enhancement. Advantages: Improves speech quality and supports robust speech analysis in noisy environments. Disadvantages: Limited datasets and model explainability remain challenges.
7. Advancements in Speech Recognition: A Systematic Review of Deep Learning Transformer Models—Sharrab et al., 2025. This work reviews transformer-based deep learning models for improving speech recognition accuracy and robustness. Advantages: Handles context, accents, noise, and enables efficient speech processing. Disadvantages: Further research is needed for improved robustness and real-time speech recognition.
8. Voice-Controlled Smart Programming Assistant—Chandu et al., 2025. This work develops a voice-based programming assistant for coding, debugging, and optimization using NLP and machine learning. Advantages: Supports hands-free coding, real-time debugging, and multiple programming languages. Disadvantages: Ambiguous commands and background noise can affect performance.

Collectively, these studies motivate EVA's focus on multimodal context, local fast-path execution, cross-session memory, and verified action outcomes.

III. PROPOSED SYSTEM

EVA is proposed as an intelligent multimodal desktop assistant that combines natural voice interaction, visual perception, persistent memory, and operating-system automation in a unified application. The system receives spoken instructions through a bidirectional audio-streaming pipeline and produces responses while maintaining an ongoing conversational session. Voice activity detection supports barge-in, allowing the user to interrupt the assistant during playback and immediately issue a new command. Screen capture and webcam input provide additional context for requests involving visible applications, documents, objects, or interface elements.

Every request is first examined by the Fast-Path Intent Router. Frequently used commands, including volume adjustment, application launching, and hardware-status queries, are matched and executed locally to minimize delay and reduce unnecessary cloud usage. Requests requiring language understanding, reasoning, or multimodal interpretation are forwarded to the cloud model. The model can issue structured function calls that are mapped to dedicated handlers for file management, browser control, system settings, document conversion, screen analysis, and application

automation. Each handler returns a structured result, and EVA reports completion only after the requested action has been successfully verified.

The proposed system also provides continuity across sessions through long-term and semantic memory. User preferences, session summaries, and relevant task information are stored locally in structured JSON records. TF–IDF and characterand word-level n-gram representations are used to retrieve semantically related records without requiring an external vector database. Retrieved information is supplied as context for later interactions, enabling EVA to remember preferences and continue earlier tasks while keeping the memory process transparent and locally manageable.

Interaction and system status are presented through a PyQt6 Heads-Up Display. The interface includes an audio visualizer, listening and speaking indicators, camera preview, system telemetry, activity logs, file-upload controls, text input, microphone control, and an emergency interruption option. A web dashboard extends selected monitoring and interaction functions to mobile devices. Safety mechanisms operate across the action layer: path containment restricts file access, isolated subprocesses reduce command-injection risk, atomic writes protect files from incomplete updates, and sensitive values are removed from logs. These mechanisms make EVA responsive and capable while preserving user control, traceability, and reliable reporting of execution outcomes.

A. Feasibility Study

Technical Feasibility: Python, PyQt6, NumPy, and the Google GenAI SDK support the required interface, streaming, memory, and multimodal functions on a conventional computer with a microphone and webcam; a GPU is optional.

Economic Feasibility: Open-source libraries and consumer hardware limit deployment cost. Local routing and retrieval reduce metered cloud usage.

Operational Feasibility: Voice control, visible status indicators, interruption controls, and verified results make the system usable without specialist knowledge.

Legal and Security Feasibility: Screen, microphone, camera, and file access require user consent. Path containment, isolated subprocesses, atomic writes, log redaction, and restrictions on unsolicited advisory content reduce privacy and execution risks.

IV. SYSTEM DESIGN AND ARCHITECTURE

A. Five-Layer Architecture

The system is organized into five cooperating layers, isolating model intelligence, presentation, orchestration, execution, and persistence/safety concerns.

B. Core Components

The event-driven core contains five components: the PyQt6 HUD and web dashboard; the Gemini Live audio stream with a 500-slice playback queue; an intent router and function-call dispatcher; action handlers for files, applications, browsers, documents, and telemetry; and a persistence layer providing JSON memory, semantic retrieval, safe subprocess execution, and redacted logging.

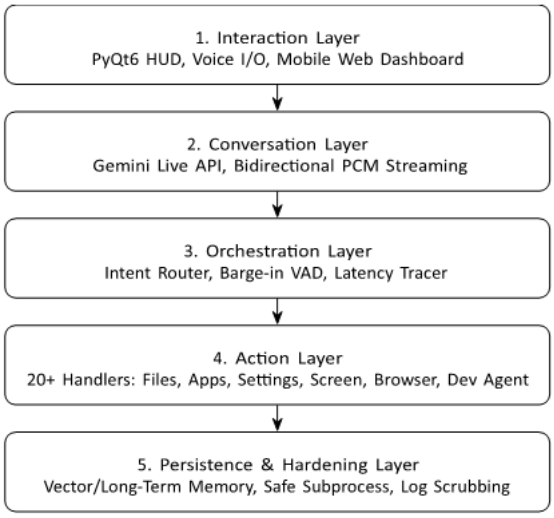

Fig. 1. Five-Layer Architecture of EVA

C. Data Design

Long-term memory is stored as structured JSON. The zerodependency retrieval-augmented generation (RAG) module in memory/vector_memory.py constructs sparse TF–IDF representations from word and character n-grams using pure Python and NumPy. For term t in memory document d, the weight is

$$w_{t,d} = \mathrm{tf}(t,d)\log\left(\frac{N+1}{\mathrm{df}(t)+1}\right), \quad (1)$$

where N is the number of stored memory records and df(t) is the document frequency. The search_vector_memory function ranks a query vector q against a memory vector d by cosine similarity:

Memory records contain a user profile, preferences, activework metadata, session summaries, and the vectors used for retrieval.

V. IMPLEMENTATION

A. Technology Stack

The implementation uses the Google GenAI SDK and Gemini Live for 16-kHz input and 24-kHz output, sounddevice for audio, PyQt6 for the HUD, mss and OpenCV for visual input, and NumPy for local semantic retrieval. Desktop actions use pyautogui, pycaw, and Playwright; FastAPI and WebSockets support remote access. Faster-Whisper/Vosk and Kokoro/EdgeTTS provide offline speech fallbacks.

B. Request Lifecycle

Figure 2 illustrates the path of a representative request: "Find my AI project presentation and convert it to PDF".

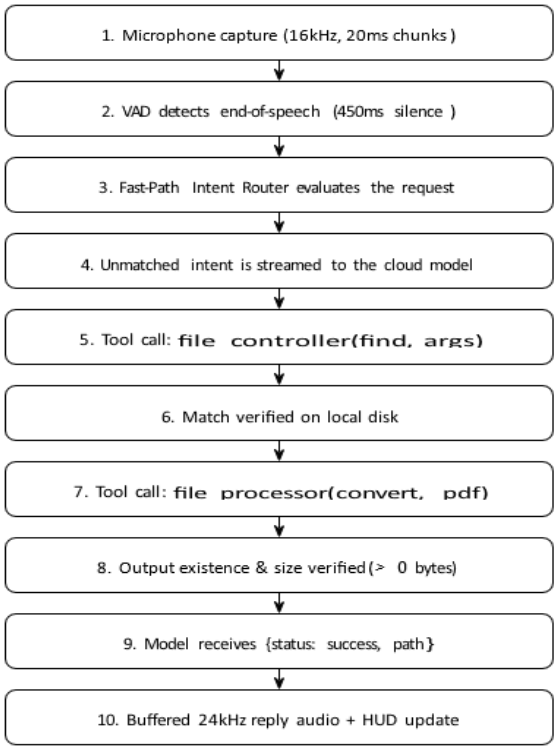

Fig. 2. End-to-End Request Lifecycle

C. Latency-Aware Execution

Each request first passes through the Fast-Path Intent Router. Common system operations are handled locally, while unmatched or complex requests are forwarded to the cloud model. Response time is measured from speech completion to the first audible output or interface update; aggregate results are reported in Table II.

D. Desktop Graphical User Interface

Figure 3 shows the PyQt6 HUD. It combines an audio-state visualizer, hardware telemetry, an activity log, a multimodal file-drop area, text input, microphone control, and an emergency barge-in button.

E. Key Implementation Details

1) Fast-Path Intent Router: core/intent_router .py resolves common operating-system requests locally before cloud fallback, producing a measured median turnaround below 30 ms.
2) Vector Memory: memory/vector_memory.py uses TF–IDF and n-grams to retrieve related records from earlier sessions.
3) Stable Audio and OCR: A 500-slice PCM queue absorbs streaming bursts, while background-window isolation excludes the HUD from OCR input.
4) Grounded Execution: Completion is reported only after a handler verifies success; subprocess arguments, timeouts, sandbox paths, and atomic writes constrain local actions.

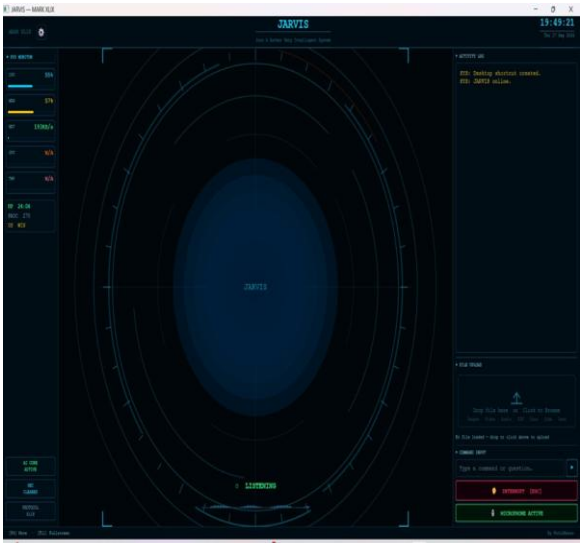

Fig. 3. EVA desktop interface prototype

5) Protected Logs: API keys, tokens, and private file content are redacted before persistence.

VI. SOFTWARE TESTING AND RESULTS

A. Testing Methodology

The Windows development system was evaluated through unit, regression, integration, black-box, and acceptance tests. All 47 pytest tests covering actions, memory, interruption, path validation, subprocess safety, and voice-to-action orchestration passed in 11.57 s. Manual tests covered the HUD, dashboard, file upload, offline fallback, concurrent telemetry clients, OCR isolation, and barge-in behavior. Latency values are aggregate development-system measurements rather than hardware-independent guarantees.

B. Test Cases

Table I

Representative Functional Test Cases

ID	Scenario	Observed Result	Status
TC-01	User interrupts active speech	Playback stopped and new input accepted	Pass
TC-02	Conversion requested for a missing file	Failure reported without false success	Pass
TC-03	Path traversal using ./../system32	Operation rejected and logged	Pass

C. Response-Time Benchmark

Response time was measured from speech completion to the first audible output or interface update. Table II shows that

Table II

Measured System Response Time and Latency Benchmarks

Task	Med.	p95	Path
Fast-path local router	< 30 ms	45 ms	Local OS API (< 1 ms)
Vector-memory RAG	3 ms	8 ms	Local vector store
Voice response	850 ms	1250 ms	Cloud stream
Screen vision	1350 ms	1800 ms	Vision model
Document/PDF	1600 ms	2400 ms	Local automation

frequent system commands bypass cloud inference and achieve a median turnaround below 30 ms.

VII. CONCLUSION AND FUTURE SCOPE

A. Conclusion

EVA integrates interruptible voice interaction, visual context, semantic memory, and verified desktop actions. Local routing achieved a median response below 30 ms, memory retrieval took 3 ms, and conversational audio began in 850 ms. Buffered playback, OCR isolation, path containment, subprocess controls, and redacted logging improve reliability and auditability. The complete 47-test suite passed in 11.57 s.

B. Future Enhancements

- Extend automation to macOS and Linux and add on-device inference.
- Introduce per-handler permissions and isolated multi-user profiles.
- Evaluate memory retrieval with larger user groups and standard benchmarks.
- Add encrypted backup for long-term memory.

REFERENCES

[1] Pabón Mendoza, K. J. Barrios Quiroga, S. D. Solano Celis, and C. M. Quintero, “NAIA: A multi-technology virtual assistant for boosting academic environments—A case study,” IEEE Access, vol. 13, pp. 141461–141483, 2025, doi: 10.1109/ACCESS.2025.3597565.

[2] R. Vora, R. K. Chakrawarti, and N. Raghuwanshi, “Machine learning-enhanced voice interaction: Revolutionizing Windows desktop applications,” in Proc. IEEE Int. Conf. Advances in Computer Science, Electrical, Electronics and Communication Technologies (CE2CT), 2025, doi: 10.1109/CE2CT64011.2025.10939579.

[3] Pundir, L. Noor, K. Gupta, and G. Nijhawan, “Development of ALEN: An advanced voice assistant with comprehensive functionalities,” in Proc. IEEE Int. Conf. Communication, Security and Artificial Intelligence (ICCSAI), 2025, doi: 10.1109/ICCSAI64074.2025.11063823.

[4] E. Fadel, A.-R. Hedar, M. Farouk, and M. Kamel, “Developing voice-based assistant machine learning techniques to enhance learning experiences for students with special needs,” in Proc. IEEE Int. Conf. Computer and Applications (ICCA), 2025, doi: 10.1109/ICCA66035.2025.11430972.

[5] K. D. Kavitha, S. Singh, A. Raj, V. Khandelwal, and M. K. Jindal, “Context-aware AI assistant using OpenAI,” in Proc. IEEE Int. Conf.

Innovative Trends in Information Technology (ICITIIT), 2025, doi: 10.1109/ICITIIT64777.2025.11040644.

[6] M. S. Remya, R. Raman, R. Sankaran, V. Namboodiri, and P. Nedungadi, "Artificial intelligence for speech classification and enhancement of speech and language disorders: Techniques, applications, and future directions," IEEE Access, vol. 13, pp. 177136–177159, 2025, doi: 10.1109/ACCESS.2025.3620114.

[7] Y. O. Sharrab, H. Attar, M. A. H. Eljinini, Y. Al-Omary, and W. E. AlMomani, "Advancements in speech recognition: A systematic review of deep learning transformer models, trends, innovations, and future directions," IEEE Access, vol. 13, pp. 46925–46950, 2025, doi: 10.1109/ACCESS.2025.3550855.

[8] D. Chandu, D. S. Sriram, and V. Ulagamuthalvi, "Voice-controlled smart programming assistant," in Proc. IEEE Int. Conf. Intelligent Sustainable Systems (ICISS), 2025, doi: 10.1109/ICISS63372.2025.11076259.